MACM 316 – Computing Assignment 7

Due Date: Friday April 3 at 11:00pm.
Submission Instructions: You must upload one .pdf file in Crowdmark that consists of two pages: page 1 is your report which should fit all discussions, data and figures into a single page; and page 2 is a listing of your code. The deadline is **11:00pm** on the due date. The actual due time is set to 11:05pm and if Crowdmark indicates that you submitted late, you will be assigned a grade of 0 on this assignment. Your TA has emailed you a Crowdmark link that you should save since it will allow you to upload your completed assignments.

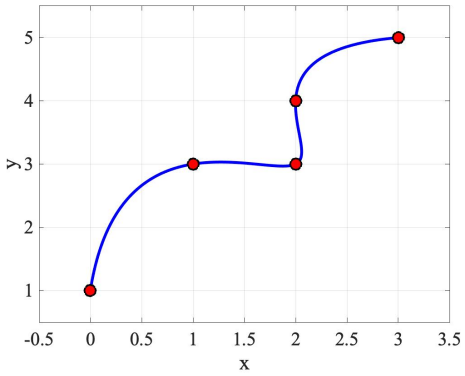
- Please review the **Guidelines for Assignments** carefully.
- Acknowledge any collaborations or assistance from colleagues/TAs/instructor.
- If you have any questions about Matlab or aspects of this assignment, then you are strongly encouraged to attend tutorials and drop-in workshops.

Parametric Splines for Multi-Valued Functions

In this assignment, your aim is to extend the definition of a cubic spline to interpolate data that cannot be described by a single-valued function.

(a) As a first "test problem", consider the data points listed in the table (below, left) which are sampled from some underlying smooth curve shown in the plot (below, right):

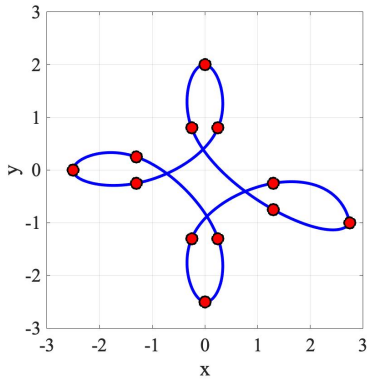
index (t)	x	y
0	0.0	1.0
1	1.0	3.0
2	2.0	3.0
3	2.0	4.0
4	3.0	5.0



Clearly, the given data have multiple y-values at $x = 2$, meaning that the underlying curve is multiply-defined – it's not a function!! This means that the usual cubic spline approach based on trying to interpolate a function of the form $y = f(x)$ will not work. The trick to handling such data is to use a parametric spline and instead treat the data as coming from a parametric curve $x = R(t)$ and $y = S(t)$ for some parameter t. In this example, it is easiest to simply assign the parameter values $t = 0, 1, 2, 3, 4$ corresponding to the index of the data points in the table. The particular choice of t values is actually not important as long as t is monotonically increasing (or decreasing – try that out if you're curious!).

Based on the t-x-y data from the table, generate two cubic splines $R(t)$ and $S(t)$ using the MATLAB `spline` function with its default not-a-knot end-point conditions. Provide plots of R versus t, S versus t, and S versus R (the last of which should reproduce the plot above).

(b) You will now apply the parametric spline idea to the more complicated curve shown below:



The 13 points that define the curve have coordinates listed below:

	0	1	2	3	4	5	6	7	8	9	10	11	12
x	2.75	1.3	-0.25	0.0	0.25	-1.3	-2.5	-1.3	0.25	0.0	-0.25	1.3	2.75
y	-1.0	-0.75	0.8	2.0	0.8	-0.25	0.0	0.25	-1.3	-2.5	-1.3	-0.25	-1.0

Using this set of ordered pairs (x, y), determine the parametric spline $x = R(t)$, $y = S(t)$ that interpolates the points using MATLAB's `spline` function. Then plot your parametric spline curve and verify (by zooming in on your plot) that the right "leaf" of your spline is different from the other three leaves in that it is not smooth – the spline endpoints meet at a cusp.

(c) For a periodic curve like the one in part (b), it is more appropriate to use the periodic end-point conditions discussed in class, $R_0'(t_0) = R_{n-1}'(t_n)$ and $R_0''(t_0) = R_{n-1}''(t_n)$ (similarly for $S(t)$) instead of the not-a-knot conditions. Use the MATLAB code `perspline.m` posted on Canvas to generate the two periodic splines approximating $R(t)$ and $S(t)$, and then compare your parametric curve to what you obtained in part (b). Verify that the cusp is eliminated.

(d) In this part, you can get creative! Start by drawing a periodic parametric curve, which can be any curve of your own design as long as the endpoints meet at the same location. Your curve should be defined by at least 20 spline points (but not more than 40) and it should have at least one location where it crosses itself (like the "leaves" in part (b)).

To generate your list of points, you may find it helpful to use MATLAB's built-in function `ginput` (graphical input from mouse) which allows you to "draw" your parametric curve on the screen and outputs the x and y coordinates for you. If you type `help ginput`, you can see that it reads mouse clicks in the plotting window until the "enter" key is hit, and then returns lists of the point coordinates. I found it best to draw my curve on a sheet of paper (the most "see-through" paper you have), place the paper over the computer screen (increase the screen brightness as much as possible), execute `ginput`, and then trace the curve on the paper, clicking on a sequence of points lying along it. You may find the following sequence of MATLAB commands helpful:

```
figure('position', get(0,'screensize')) % biggest window possible
axes('position', [0 0 1 1]) % domain [0,1] x [0,1]
axis square                 % x,y axes equal
[x,y] = ginput;             % record mouse clicks
close                       % get rid of huge window
save mydatafile.mat x y     % save the data points
```

The `save` command saves your data points to a file that can later on be read back in using the `load` command. You may then use your x, y data as input to `perspline.m` in the same way you did in part (c) to construct the periodic spline and plot your results.